(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 549 140 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **92310801.3**

(51) Int. Cl.^{5}: **G06F 15/403**

(22) Date of filing: **25.11.92**

(30) Priority: **23.12.91 US 812677**

(43) Date of publication of application:
30.06.93 Bulletin 93/26

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Strickland, Jimmy Paul**
18929 Alcott Way
Saratoga, California 95070(US)
Inventor: **Kapulka, Kenneth Michael**
6378 Felder Drive
San Jose, California 95123(US)

(74) Representative: **Burt, Roger James, Dr.**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) Record updating method.

(57) A Virtual Storage Access Management (VSAM) technique for maintaining data integrity of the record-level shared data in a multiprocessor system environment without the concurrency loss associated with locking at the data Control Interval (CI) or block level. Two or more concurrent processes can update different records within the same VSAM data CI using only a record-level lock. This feature relies upon multiple private data CI copies, Shared External Storage (SES), cache cross-invalidation, and a conditional writing operation. A first process updates a private copy of a data CI containing the target record from SES, logs the changes, and writes the changed data CI back to the SES conditioned upon no update having been made to the same block data CI by another process in the interim. If a second process has indeed updated the same data CI in the interim, the first process recycles by obtaining the most recent data CI copy from SES, combining it with the first process update log entries and attempting to again conditionally write the modified data CI to SES. This process of logging updates and spinning on an intervening update by another process preserves serialization by record-level locking alone.

This invention relates to database integrity management procedures for multiple concurrent transaction processing systems, and more particularly, to a method for concurrent record updating within a single data Control Interval or block.

The lock manager portion of an operating system in a multiprocessing environment assigns and reassigns locks on storage resources on behalf of processes. This locking activity ensures proper transaction serialization and database integrity, but may lead to a decrease in process concurrency. Data processing throughput can fall when the lock manager frequently locks a resource having a high concurrency requirement arising from many concurrent processes operating with the same resource. This lock contention situation requires the lock manager to allocate such a resource among the concurrent processes. Moreover, many of the processes are slowed by the wait states imposed to sort out the lock contention. The lock manager must also consume processing time to lock, unlock and resolve wait and resume conditions.

Because a relational database management system operates with concurrent processes in a multiprocessor database environment, it uses facilities such as lock managers. This can be appreciated by reference to C.J. Date, "A Guide to DB2", Addison-Wesley Publishing Co., pp. 191-195 (1984); and C.J. Date, "An Introduction to Database Systems", Addison-Wesley Publishing Co., pp. 422-427 (1986). H.F. Korth, et al, "Database System Concepts", McGraw Hill, Inc., pp. 356-402 (1986), gives a general discussion of concurrency control in database systems.

In multiprocessing systems generally, and database management systems in particular, when one user accesses a file for editing or updating purposes, all other users are locked out until the accessing update is completed or committed. To improve concurrency in a shared file environment, multiple users should be permitted to read a file that is being concurrently updated. Also, one user should be permitted to update records in the same file in which another user is updating different records.

USA 4,716,528, discloses a method that typifies such hierarchical locking and lock promotion techniques with the variable granularity that can manage the trade-off between lock processing overhead and concurrency. This, a coordinated pair of locking limits where a first limit is placed on the number of "small granularity" locks per resource and a second limit is placed on the total number of locks assignable to each process. When the "small" lock limit is breached, the method withdraws all small locks and grants a single global lock to the entire resource (lock escalation), thereby reducing the total number of locks. When the process requests an additional lock over the total lock number limit, the lock is refused. There is no suggestion of a technique of constant granularity (and, thus, low overhead) for avoiding lock contention at the block level during concurrent block access.

US-A-5,043,876 discloses an N-level file shadowing technique for use in a shared file environment. N-level shadow copies of each shared file are maintained to allow multiple users to read a consistent copy of the file even though other users may be simultaneously updating the same file. Every reader who opens the shared file sees the latest committed copy of the file and is unaware of updating transactions occurring after the file is opened. However, this permits only one updating transaction at a time, forcing other concurrent updating processes to wait because of update lock contention at the file or block level.

Although the locking contention and concurrency problem is keenly felt, no suggestion has been recorded for managing the concurrent accession of common data blocks containing multiple records in shared memory by record-level locking alone. That is, access by two or more concurrent updating transactions to the same block or data Control Interval (CI) normally requires locks at the data CI level. There is a strongly-felt need in the art for a method permitting concurrent access to a single block by two or more updating processes with serialization by record-level locking alone. The related unresolved problems and deficiencies are keenly felt and are solved by the present invention in a manner described below, avoiding lock contention at the block level.

The following acronyms are used herein as defined below and may be appreciated with reference to "MVS/ESA VSAM Administration Guide", SC26-4518, International Business Machines, Inc., Armonk, NY, and Marilyn Bohl, "Introduction to IBM Direct Access Storage Devices", Science Research Associated, Inc., Palo Alto, CA (1981):

API	Application Programming Interface
BMF	Buffer Management Facility
CA	Control Area
CI	Control Interval
CI/CA	Control Interval/Control Area
CIDF	CI Definition Field
KSDS	Key-Sequenced Data Set
PB	Private Buffer
RLS	Record Level Sharing
RMR	Record Merge Redo
RPL	Request Parameter List
SDSH	Shared Data Storage Hierarchy
SES	Structured External Storage
SLC	Shared Local Cache
TLCE	Test Local Cache Entry
UOW	Unit Of Work

The methods of this invention permit a Virtual Storage Access Method (VSAM) to maintain record-level shared data integrity in a distributed multiprocessor environment without the overhead of block or data Control Interval (CI) level locking. Two or more processes may concurrently update different records in the same CI, using only a record-level lock. This invention includes methods for multiple Private Buffer (PB) data CI copies, a Shared Local Cache (SLC) invalidation protocol and a conditional write function.

In operation, the VSAM Record Level Sharing (RLS) methods of this invention permit a first process or Unit Of Work (UOW) to fetch a Private Buffer (PB) copy of a data CI containing the record of interest from shared Structured External Storage (SES) to a PB in Shared Local Cache (SLC). After the fetch, the first UOW logs all updates to the record of interest and conditionally writes the changed data CI back to SES. The SES write is conditioned upon there not having been an interim write to the same data CI by a second UOW. If a second UOW has indeed written the same data CI in the interim, the first UOW recycles by referring to SES for the most recent copy of the data CI, combining it with the logged updates from the first UOW and again conditionally writing the updated data CI back to SES.

Thus, the RLS methods of this invention do not require Shared Local Cache (SLC) buffer or data CI level locking for serialization of changes to shared data sets nor for SLC consistency. Instead, the novel protocols of this invention for testing cache entry validity and for conditionally writing to cache are used to detect data CI level contention.

It is an object of this invention to preserve updating process serialization by record-level locking alone. This objective is met by the Private Buffer (PB) copy, the SLC buffer invalidation and the conditional SES write methods of this invention.

It is an advantage of these methods that they avoid the overhead of locking and unlocking data CIs, which is a significant overhead cost in multi-system data sharing environments. It is another advantage of this invention that the new protocols reduce data sharing lock contention for multiple updating transactions accessing different records within the same data CI.

It is yet another advantage that UOWs may read unmodified records from a data CI that contains records concurrently accessed by updating transactions without waiting for the updating transactions to commit. That is, record-level read/write operations do not encounter data CI (block) level contention.

The methods of this invention can be appreciated as a hybrid locking and versioning technique. Each Private Buffer copy in Shared Local Cache represents a consistent but possible outdated data CI version accessible only by the Unit Of Work for which it was created. Record-level locking is employed.

The absence of CI level locking permits data CI copies to exist concurrently in more than one Shared Local Cache. Each of these SLC copies may be updated concurrently, and the "validity" of each SLC copy is monitored at the SES level. Record-level locking is performed, so any concurrent updates within the same data CI must be against different records. Where concurrent updates are made to different copies of the same data CI, the multiple SLC copies are then merged according to the Record Merge Redo (RMR) method of this invention to form a new data CI copy in SES containing all updated records from the several concurrent transactions.

The Unit Of Work (UOW) is a new parameter of this invention that must be specified on the VSAM Request Parameter List (RPL). VSAM maintains an internal structure to track all activity on behalf of a single UOW. Included in the UOW structure is a pointer to a Private Buffer in SLC containing the associated PB copy of the data CI and a UOW log block listing the associated data CI change log for the UOW.

Only a single PB is maintained per UOW per data set. When the UOW accesses a second data CI, the first data CI is released either by an unconditional release if unchanged or by the conditional write method of this invention. The conditional write request results in a cycle of testing, invalidation, merging and reattempted conditional write until a new (updated and merged) data CI is successfully written to SES.

A special data CI level locking method is used to overcome the problems presented by VSAM data Control Interval and data Control Area splitting (CI/CA splitting) procedures that occur in response to occassional exhaustion of existing space within a data Control Interval or Control Area. These splitting procedures can be appreciated with reference to the VSAM Administration Guide or the Bohl article cited hereinbefore.

The scope of this invention is defined by the appended claims; and how it can be carried into effect is hereinafter particularly described with reference to the accompanying drawings in which:-

Figures 1A and 1B show the Shared Local Cache (SLC) and Structured External Storage (SES) organization of the data CI and VSAM UOW information;

Figure 2 shows the SLC buffer organization required for the VSAM Record Level Sharing (RLS) invention;

Figure 3 illustrates the Record Merge Redo (RMR) method of this invention applied to two

SLC Private Buffer (PB) copies of a data Control Interval (CI);
Figures 4A to 4D show the evolution of SLC contents during an RMR example;
Figures 5A and 5B shows the evolution of SLC contents for a KSDS CI/CA split and index update example;
Figures 6A and 6B show the effects of the data CI movements and index updates performed during a Control Area (CA) split operation; and
Figures 7A to 7D show the SLC evolution during an exemplary RMR operation conducted during a CI split operation.

The Virtual Storage Access Method (VSAM):

The methods of this invention are discussed in terms of the VSAM protocol used by IBM. VSAM is a facility for managing space in a hierarchical storage facility in a manner well-known in the art and is used herein to illustrate the operation of the methods of this invention. The references cited above provide all VSAM details necessary to appreciate the VSAM application of this invention. This invention is not limited to VSAM applications and relates to any similar distributed concurrent transaction processing systems.

In an exemplary multiprocessor storage hierarchy Figure 1A a Structured External Storage (SES) facility 10 is coupled to a plurality of Shared Local Caches (SLCs) 12. Data stored in SES 10 is typically organized in data Control Intervals (CIs) 14 of records. Each data CI is a fixed-length area of secondary storage containing records of some particular type of data. Examples of such records include Key-Sequenced Data Set (KSDS) records that are ordered in key sequence within the storage. A plurality of data CIs is typically organized in fixed-length regions of secondary storage called data Control Areas (CAs) (not shown).

A data CI is a continuous region of secondary storage that VSAM uses for storing data records and the control information describing them. It is the typical unit of data transfer between one storage level and another in a storage hierarchy. Its size varies from one data set to another, but the size of each data CI is fixed within one data set, either by VSAM or by the user within limits acceptable to VSAM. VSAM chooses the size based on the type of secondary storage hardware in use, the size of user data records, and the smallest amount of virtual storage space that user application programs make available for I/O buffers. Thus, a data CI is a device-independent unit of data transfer and may generally either span or be spanned by other data storage measures such as tracks, blocks or records.

In some instances, an insertion or a lengthening of a record will require more free space than is available within the corresponding data CI, leading to a "Control Interval split" operation wherein VSAM moves some of the stored records in the data CI to an unused data CI in the same CA. This frees some space to hold the new or updated records.

If no free data CI exists in the CA, an insertion requiring a free data CI leads to a "Control Area split" wherein VSAM establishes a new Control Area, either by using space already allocated or by extending the data set if the initially allocated space is full and extensions are provided for in the data set definition. VSAM then moves the contents of about half the data CIs in the full CA to the free data CIs in a new CA. It then completes the triggering insertion operation by inserting the new records into the one of the two CAs dictated by the record key. Since about half the data CIs in each of these CAs are now free, subsequent insertions should not require additional CA splitting. CI/CA splitting is not a common occurrence for data sets with sufficient distributed free space and occurs automatically under VSAM control.

The VSAM Record Level Sharing (RLS) Invention:

The Record Level Sharing (RLS) methods of this invention extend the VSAM Application Programming Interface (API) to include support for application Units Of Work (UOWs). UOW is a new parameter specified by way of the VSAM Request Parameter List (RPL). VSAM maintains an internal structure through which it tracks its activity on behalf of a UOW. A UOW block (Figure 1B) includes a pointer 16 to a copy of a data CI in a Private Buffer (PB) obtained from the VSAM Buffer Management Facility (BMF) and a pointer 18 to the UOW log containing information about any record modifications made to the Private Buffer (PB) copy 20 by PUT/ERASE requests that specify the UOW in their RPLs.

VSAM maintains information about no more than one BMF buffer per UOW per data set. In general, VSAM maintains the Private Buffer for the UOW until the UOW requests a reference to a different data CI within the data set. VSAM then issues a RELEASE NOWRITE request to BMF to release an unmodified buffer and a RELEASE WRITE request to write and release a modified buffer.

The BMF manages all the buffers and Figure 2 illustrates the relationship between the UOW Private Buffers and the SLC 12. The SLC is a storage level in the Shared Data Storage Hierarchy (SDSH). Within the SLC, a number of data CI copies e.g. 22, are stored, having been written from

SES 10 (Figure 1A). SES 10 may write other copies to other SLCs but must monitor the "validity" of all such local CI copies. The UOW Private Buffers 20 (Figure 1B) may also be contained in SLC and are used to hold record updates made by single UOWs.

The local copy of data CI 22 (Figure 2) can be written into a UOW Private Buffer as many times as necessary, depending on the number of UOWs that concurrently access the local copy of data CI 22. Thus the UOW Private Buffer PB24 is a first copy for UOW i, and PB 26 is a second copy of data CI 22 for UOW j.

After UOW i requests access to a second data CI 28, the VSAM issues one of two available RELEASE instructions for PB 24. The RELEASE NOWRITE is an unconditional release of an unmodified PB. The RELEASE WRITE is a conditional write request that fails when PB 24 or data CI 22 has been invalidated by a RELEASE WRITE issued to another copy of the same data CI on behalf of another sharing UOW, either locally or in another SLC. This happens when two or more UOWs obtain copies (through LOCATE UPDATE commands) of the same data CI, modify the buffers, and then attempt to write (through the RELEASE WRITE command) the modified Private Buffers. The first RELEASE WRITE is successful. The others fail. The failing writes caused VSAM to perform the Record Merge Redo (RMR) processing described hereafter.

VSAM uses the following new functions to obtain cached access to data CIs of record-level shared VSAM data sets.

(a) LOCATE READ Function: This function opens UOW access to a shared buffer in SLC, such as data CI 22 (Figure 2). If the requested data CI is not available in the local SLC, VSAM causes a local copy to be written from SES 10 (Figure 1A). Multiple local UOWs may share (read) shared buffer 22.

(b) LOCATE UPDATE Function: This function opens UOW access to a Private (non-shared) Buffer copy of a data CI, such as PB 24 (Figure 2). This function also creates a consistency or "update version" token and writes it to the PB content. The PB copy of the data CI is created in response to a request for access by an updating UOW and is abolished (released) in response to a request for access to another data CI by the same UOW.

(c) RELEASE NOWRITE Function: This function removes a calling UOW's access to a PB copy of a data CI. This function is invalid for any PB copies that have been updated or changed. The subject PB copy is abolished or released.

(d) RELEASE WRITE Function: This is the conditional write function that writes the PB contents to the data CI SLC buffer and to the central SES cache. The first step is to check the PB consistency token for validity and to return a conditional write failure message if the token is invalid. If the token is valid, the next step is to write the PB content to the SLC buffer (e.g. buffer 22) and to request permission from SES cache to write the PB content to SES. If SES has invalidated the SLC buffer copy of this data CI (through the SES CROSS-INVALIDATE subfunction of an earlier RELEASE WRITE operation), the next step returns a conditional write failure message. If the SES cache accepts the PB contents, this function first generates and writes a new consistency token to the PB that invalidates all previous consistency tokens and then writes the new consistency token to the SLC shared-buffer. Finally, the updated SLC buffer is then written to SES, which then invalidates all other SLC copies of the data CI with the SES CROSS-INVALIDATE subfunction. Upon successful transfer of PB contents to both SES and SLC, the UOW access to this PB is removed, abolishing the PB.

(e) TEST BUFFER VALIDITY Function: This function checks the local consistency token in SLC buffer and issues a MVS Test Local Cache Entry (TLCE) Validity command, which checks SES to ensure that a SES CROSS-INVALIDATE operation has not yet occurred.

(f) SES CROSS-INVALIDATE Function: This is a SES cache subfunction of the RELEASE WRITE function and is the basis for SLC consistency in the multi-system environment. The SES registers and monitors the contents of a set of connected SLCs. When a SLC entry is written to SES, the SES invalidates any existing copies of the named data CI in other SLCs, causing any subsequent conditional writes to SES of the invalidated SLC entries to fail. This conditional write capability of the SES cache is the means for determining write contention across the SLC boundaries on a VSAM data CI. Failure of the conditional write triggers the VSAM RMR process.

(g) Test Local Cache Entry (TLCE) Validity Function: This is a standard function provided by the DFP BMF and the SES cache. The VSAM RLS methods of this invention use the BMF TLCE function in instances where a VSAM record must first be located before locking. Such instances include those where application requests do not specify the complete record identifier. Examples are GET sequential, GET key greater than or equal, and GET generic key search. For such requests, VSAM uses BMF to obtain access to SLC buffers containing data CIs. The CIs are searched to locate the requested record. Having located the record, VSAM knows the complete record identifier (the key for KSDS or the RIN for key-less types). A record lock is then granted using the record identifier as the lock name. Before returning the locked record to the application, VSAM must ensure that the SLC copy of the record is the most current version of the record as determined by the central data in SES cache. VSAM uses the BMF TLCE Validity function to verify that the accessed data CI has not been changed by some other UOW in some SLC. The combination of the two requirements for SLC buffer validity and for exclusive record lock ensures that the most current version of the record is being returned to the application. If the SLC buffer copy is not valid or if the record is not found locally, VSAM calls BMF to obtain a new buffer copy in SLC of a SES version of the data CI. Also, for a KSDS record, a CI/CA split operation may have moved the desired record to a new data CI. In such a case, VSAM searches the KSDS index to locate the new data CI actually required.

(h) Record-Level Locking Function: The VSAM RLS methods of this invention do not use SLC buffer entry (data CI) level locking. Instead, the TLCE Validity and the RELEASE WRITE functions are used to detect SLC buffer entry level contention. This approach avoids the SLC buffer locking overhead, giving significant savings for distributed data sharing systems. This approach also reduces lock contention because multiple transactions may update different records within the same data CI without waiting for one another. Also, unmodified records can be read from copies of a data CI that contains records modified by other UOWs without waiting for commit. Thus, record operations do not encounter SLC buffer level contention. The absence of data CI level locking permits the situation described above where copies of a data CI may reside concurrently in more than one SLC. Each of these SLC copies may be concurrently updated. Record locking is performed so that any concurrent updates are always made against different records within the data CI. Such concurrent updates leads to a requirement for the RMR function.

(i) Record Merge Redo (RMR) Function: This function merges the changed records in the multiple data CI copies from the various SLCs to form a single data CI copy containing all of the changed records. This can best be appreciated through the following example.

A data CI copy CI 1 (Figure 3) in SLC 1 has Record 3 changed. Another copy of data CI 1 in SLC 2 has Record 14 changed. The LOCATE UPDATE function can generate such multiple PB copies of the same data CI within the environment of a single SLC. Thus, whether two such PB copies of data CI 1 exist in the same SLC or in different SLCs, this function permits merger of these two PB copies to form the single Record Merged Copy of CI 1 with Records 3 and 14 changed.

During the RMR operation (Figures 4A to 4D) the SLC and SES copies of a data CI change. System 1 and System 2 both update records within the sane data CI 1. Each system first reads a copy of CI 1 into its SLC and updates a record in the SLC copies of CI 1 (Figure 4A).

System 1 next issues a RELEASE WRITE (conditional write) command for its copy of CI 1. The RELEASE WRITE is successful and invalidates all other SLC copies of CI 1 including the SLC entry in System 2 (Figure 4B).

When System 2 issues a RELEASE WRITE command for its copy of CI 1, the conditional write fails because the System 2 SLC entry was invalidated by the earlier successful RELEASE WRITE of the System 1 copy. Upon notification of the failure of its RELEASE WRITE command, System 2 then issues a SES read command to obtain a valid copy of CI 1. After receipt of the valid copy, System 2 now has two different copies of CI 1 (as illustrated in) (Figure 4C).

VSAM then reapplies the changes to Record 14 imposed by the System 1 UOW to the valid copy of CI 1 obtained from SES. System 2 reissues the RELEASE WRITE for the newly updated valid copy of CI 1 and this write is successful. The RMR operation is now complete (Figure 4D). Because of the latest successful SES write by System 2, the SLC copy of CI 1 in System 1 (Figure 4B) is now invalid.

The CI/CA Split Problem:

The several RLS functions described hereinafter must be modified slightly to adapt to the problems presented by the VSAM CI/CA split function. The methods of this invention use a global data set level exclusive lock to serialize CI/CA splits and the related index updates for a data set. The index and data changes made by a CI/CA split are written to the SLC and SES caches immediately. The changes are not buffered in UOW Private Buffers.

Although CI/CA splits are serialized for a data set, they are not serialized with respect to the Get/Put/Erase requests that do not cause CI/CA splits. The record-level lock of this invention does not inhibit a CI/CA split of the CI/CA containing the locked record. Also, the existence of PB copies of a modified CI do not inhibit another UOW from splitting the same data CI or splitting the CA containing the same data CI. Thus, some coordination and serialization across CI/CA splits and the concurrent access and deferred write of CIs involved in the split is required.

The movement of existing records from one CI to another during a VSAM CI/CA split demands some extension to the VSAM RLS serialization methods of this invention. Since the RLS record lock name is based on the record key, movement of a record from one CI to another does not affect the record lock, and the record lock properly serializes access to the record before, during and after the CI/CA split. However, the SES CROSS-INVALIDATE function of this invention is not in itself sufficient to maintain data CI integrity when the CI/CA split operation moves records to new CIs. The following example illustrates a typical problem with serialization during CI/CA splits.

UOW 1 (Figure 5A) has changed Record 2 in its copy of data CI 1 in a UOW 1 PB. It has not written its copy of CI 1 to either SLC or SES.

UOW 2 next causes a split to CI 1 by some operation that exceeds available CI 1 space in UOW 2 PB. The CI split uses two SLC buffers and forms CI 1 and CI 8 (Figure 5A). The UOW 2 copy of CI 1 does not contain the UOW 1 changes to Record 2.

The CI split data movement and index update steps for UOW 2 are: (a) create and write CI 8; (b) update the index to change the key for CI 1 and add an entry for CI 8; and (c) write a new CI 1 removing the records that were moved to CI 8.

Now suppose that, between the index update and new CI 1 write steps of the UOW 2 CI split, a UOW 3 searches the index, locates CI 8, modifies Record 15 and writes an updated CI 8. The result is a buffer situation for all three UOWs as shown in Figure 5A.

Now, UOW 1 writes its PB copy of CI 1. UOW 2 has not yet written its PB copy of CI 1. Hence, the UOW 1 PB copy is still valid and the UOW 1 RELEASE WRITE to SLC and SES (not shown) is successful. This write invalidates (by an SES CROSS-INVALIDATE) the UOW 2 PB copy of CI 1. When UOW 2 now attempts to write CI 1, the conditional write fails.

The situation created in this example and illustrated in Figure 5A is far more complex than the RMR example discussed above in connection with Figure 4. UOW 2 cannot access the log of record changes made by UOW 1 to CI 1 because the log originally linked to UOW 1 was deleted upon the successful RELEASE WRITE of UOW 1. Also, the logged record changes to CI 8 by UOW 3 must be saved somehow.

In a variation on the example discussed above in connection with Figure 5A, . UOW 1 (Figure 5B) holds a PB copy of CI 1 in which it has changed Record 10. The UOW 1 PB copy has not been written to SLC or SES.

UOW 2 then causes a split to CI 1 in some fashion. During this split, UOW 1 successfully RELEASE WRITES its PB copy of CI 1 and releases its lock on Record 10. The split has already moved the old version of Record 10 to the new CI 8 and has updated the KSDS index showing the new key location.

UOW 3 then obtains a lock on Record 10 and locates (by way of new KSDS index) the old version of Record 10 held in the new CI 8. This action violates the basic record accession integrity of the Record Level Sharing method of this invention and must be prevented.

This invention uses a preferred serialization technique to coordinate record/CI accession with CI/CA split processing rather than incorporating an alternative complex RMR procedure to handle all of the various situations that might occur. These preferred CI/CA split serialization techniques are now described hereinafter.

The CI split process begins by setting a bit (CIDFBUSY) in the Control Interval Definition Field (CIDF) and then writing the CI. This announces to all accessing processes that the CI split is in progress. Should a system fail before the CI split is complete, all future accesses to the CI note the set CIDFBUSY bit and invoke the necessary recovery processing to clean up or to complete the split. The last step in a CI split is to rewrite the CI that was split absent the records that were moved and with the CIDFBUSY bit disabled.

For the RLS method of this invention, the RELEASE WRITE function that normally is used to set the CIDFBUSY bit may fail because of SLC invalidation. The CI split process must then ask BMF for a valid copy of the CI and must repeatedly attempt to set the CIDFBUSY bit until the bit stays set.

The RLS method of this invention permits a CI split to finish once it has set the CIDFBUSY bit without interference from other UOWs attempting to write the CI to SLC or SES. The successful write that sets CIDFBUSY in the splitting CI triggers an SES CROSS-INVALIDATE operation that invalidates any other existing UOW PB copies of that CI. If a UOW modifies (or has already modified) its PB copy of the data CI and then attempts to write the invalidated PB, the conditional write will fail. The UOW then obtains a valid copy of the CI from BMF and finds the CIDFBUSY bit set. In response to this set bit, the UOW is obliged to request the "CI split lock". This split lock operates to serialize the UOW with the CI split process and is the preferred method for doing so.

Thus, when a UOW reads a CI and sees a set CIDFBUSY bit, it immediately requests the CI split lock. This UOW waits for the completion of the CI split process and release of the CI split lock.

The situation where a UOW has a PB copy of a data CI when activity by another UOW splits the data CI can be summarized as follows.

(a) If the UOW PB copy of the CI was obtained during the CI split, it contains the CIDFBUSY bit set. This set bit causes the UOW processing to release the PB and to request the CI split lock.

(b) If the UOW PB copy was obtained before the CI split, the CI split step of setting the CIDF-BUSY bit has already operated (by SES CROSS-INVALIDATE) to invalidate the UOW PB copy. Subsequent processing of requests for that UOW will detect the invalid PB copy as follows:

(1) Sequential Processing: After obtaining a lock on the record and before returning the record to the application, VSAM issues a TEST BUFFER VALIDITY request to BMF. The request fails.

(2) PUT-for-Update Or ERASE Functions: The subject record was located and locked by a previous GET-for-Update request. Thus, the record may be changed in the PB holding the invalid data CI. Later, when VSAM attempts to write the invalid PB copy, the BMF RELEASE WRITE functions fails on PB copy invalidity.

(3) Direct or Skip Sequential Request: Before attempting to locate the requested record, VSAM releases the UOW PB. If the PB was modified, a BMF RELEASE WRITE is issued. The request fails because of SES invalidation.

Whenever a TEST BUFFER VALIDITY function fails and the PB was modified and whenever a RELEASE WRITE function fails, VSAM performs a RMR function to reapply the record modifications to the data set. Thus, the combination of PB invalidation, the actions taken when the CIDFBUSY bit is set and the CI split lock together provide serialization between CI split operations and concurrent record updates to the same data CI. This preferred method avoids the two problems discussed above in connection with Figures 5A and 5B as follows.

In the example illustrated in Figure 5A, UOW 1 wrote CI 1 after UOW 2 split CI 1. In the interim, UOW 3 changed and wrote the new CI 8 created by this CI split. When UOW 2 tried to write CI 1, the write failed because the UOW 1 write had invalidated the UOW 2 PB copy. UOW 2 was not able to recover from this situation because it cannot determine which records were changed by UOW 1 and UOW 3. Moreover, the record locks may have already been released by UOW1 and/or UOW 3.

This problem is solved by the above CIDF-BUSY based serialization procedures. The CI split process for UOW 2 sets the CIDFBUSY bit and writes CI 1, invalidating the UOW 1 PB copy, before performing the data movement steps. This inhibits UOW 1 from writing back its copy of CI 1 until the CI split completes because, if UOW 1 attempts the write before the CI split completes, the write fails for UOW 1 PB invalidity. UOW 1 then begins a RMR procedure to revalidate its PB copy and, if UOW 1 rereads CI 1 before the CI split is complete, the CIDFBUSY bit is found to be set and UOW 1 requests and waits for release of the CI split lock in the manner discussed above.

In the example illustrated in Figure 5B, the CIDFBUSY based serialization procedure keeps UOW 1 from writing the updated version of Record 10 and UOW 1 thus retains the record-level lock on Record 10 until the CI split completes. UOW 3 waits on the record lock until UOW 1 completes the update, which in turn waits for the CI split to complete. Record access integrity is thereby maintained.

In the CI data movements and index updates performed by a CA split operation, the CA (Figure 6A) before the CA split operation includes four CIs.

The CA split operation (Figure 6B) moves the contents of CI 3 to CI 5 and the contents of CI 4 to CI 6. The split updates the KSDS index 30 to reflect the new CA and the data movement. CI 3 and CI 4 are rewritten as free space (empty) CIs. Two CAs (Figure 6B) result from the CA split operation.

CA split data movement done concurrently with record updates by other UOWs to CIs in the CA can create complex RMR requirements similar to those illustrated in the CI split examples. Thus, the writing of modified CIs is serialised with the CA movement of those CIs. Three serialisation methods need to be considered.

(1) Buffer Invalidation: This approach obtains and holds a buffer for each CI that is being moved by the CA split operation to a new CA. This permits the CA split to detect an update to a CI that was moved. An attempt by the CA split to rewrite the moved CI with a free space CI then fails because of PB copy invalidation. Thus, the CA split operation must reread the CI and then rewrite it to its new location in the new CA. Unfortunately, another UOW may have modified and rewritten the CI during the interim when the CI was moved to the new CA. This is a similar situation to the one discussed above in connection with Figure 5A. The CA split operation must merge the changes made by the other two UOWs and no log of these changes is available. Another problem with this approach is that the system that is performing the CA split may fail during the split operation. Such a failure would remove all invalidation indications from the CA

split buffers. It is not clear how recovery from such a situation can be performed. Accordingly, this approach to CA split serialization is not adopted.

(2) CI Level Locking During the CA Split Operation: This approach requires VSAM to inform BMF of the set of CIs that will be moved during the CA split operation. BMF must then either use cross-system signalling directly or use some new DFP lock manager function (not currently identified as a requirement and thus not defined) to inhibit access to all buffers containing the CIs that are to be moved by the CA split operations. When VSAM requests access to one of the CIs and receives such an "access-inhibited response", a request for the CI split lock is issued to ensure that all processing waits for completion of CA split operation. This technique seems to work but it requires cross-system signalling to start and stop the CI level serialization and the necessary logic is exceedingly complex.

(3) Set CIDFBUSY Bit in All CIs to be Moved: Prior to moving any CIs or updating the index, the CA split operation sets CIDFBUSY in each CI that will be moved. This approach has the advantage that it is the same serialization technique used for the CI split situation discussed above in connection with Figure 5A and 5B. Thus, it is the preferred method for both CI split and CA split serialization.

RMR/CI Split Example:

In a RMR operation during a CI split operation, UOW 1 has changed two records in its PB copy of data CI 1 (Figure 7A). Before UOW 1 writes its modified PB copy, UOW 2 splits this CI (Figure 7B). The CI split leaves one of the records (old copy) modified by UOW 1 in the original CI 1 (Figure 7C) and moves the other one to the new CI 8. The UOW 1 RMR operation accesses and changes both CI 1 and CI 8 (Figure 7D).

UOW 1 has changed Record 3 (Figure 7A) and Record 15 in CI 1. UOW 1 has not yet written the modified PB copy of CI 1.

When UOW 2 splits CI 1 (Figure 7B) because UOW 1 has not yet written the modified PB copy of CI 1, the CIs written by UOW 1 contain copies of Record 3 and Record 15 as they were before modification by UOW 1. The CI split operation successfully writes CI 1 and CI 8 and then updates the KSDS index.

Following completion of CI split, UOW 1 attempts to write CI 1. Because the UOW 1 PB copy of CI 1 was invalidated by the CI 1 write made by the CI split, UOW 1 begins the RMR. The RMR operation obtains a valid copy of CI 1 (Figure 7C) and reapplies the logged updates by UOW 1 for Record 3.

The RMR operation detects the change in key range of CI 1 and notes that this key range no longer includes Record 15. The valid PB copy of CI 1 is then written and released through the RELEASE WRITE function.

The KSDS index is next searched to locate Record 15 in CI 8 and the logged changes by UOW 1 to Record 15 are then applied to CI 8 (Figure 7D).

The PB copy of CI 8 (Figure 7D) is then written and released through the RELEASE WRITE function. The PB copy of the obsolete version of CI 1 is released through the RELEASE NOWRITE function, and the RMR operation is complete.

Claims

1. A method for updating a first record in a concurrent transaction processing system where a plurality of asynchronous processes access records in a data set organized as a plurality of data Control Intervals (CIs), said access being through a shared Structured External Storage (SES) means for storing the data set, comprising the steps of:

    (1) identifying the SES data CI containing the first record in response to an update access by a requesting process;

    (2) creating a Private Buffer (PB) CI copy of the latest version of the SES data CI that is accessible only by the requesting process;

    (3) updating the first record in the PB CI copy according to the requesting process to form an updated PB CI copy;

    (4) conditionally writing the updated PB CI copy to the SES data CI if said latest version of the SES data CI has not yet been updated by another process, otherwise repeating steps (2) to (4); and

    (5) terminating access by the requesting process to the PB CI copy.

2. A method according to Claim 1, comprising the additional steps of:

    (2.1) granting an exclusive record-level lock to the requesting process on the first record; and

    (2.2) adding to the PB CI copy consistency token means for determining data validity.

3. A method accordingly to Claim 2, wherein the conditional writing step (4) comprises the steps of:

    (a) testing the validity of the consistency token means and repeating steps (2) to (4) until the consistency token means is found to be valid;

(b) writing the updated PP CI copy to the SES data CI; and
(c) revising the consistency token means in the updated PB CI copy to invalidate all earlier consistency token means.

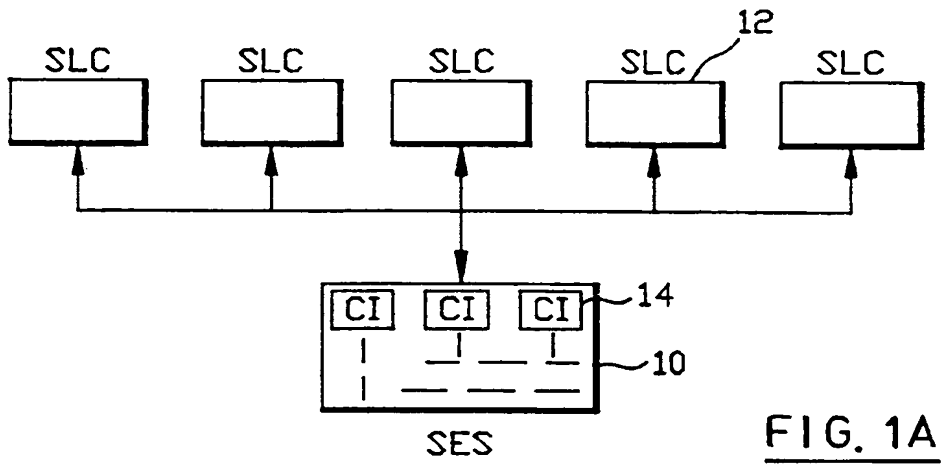
4. A method accordingly to Claim 3, comprising the additional step of:
writing the PB CI copy to a Shared Local Cache (SLC), whereby all other PB copies of the SES data CI are invalidated.

5. A method as claimed in Claim 2, 3 or 4, comprising the additional steps of:
setting a CI busy indicator means for announcing pending or actual movement in each data CI that contains records that are subject to movement by a VSAM CI/CA split process that splits data CIs and data Control Areas (CAs) into at least two new data CIs or CAs; and
granting an exclusive lock to the requesting process on the data set in response to access by the requesting process to a data CI having the CI busy indicator means set, whereby the requesting process is serialized with the VSAM CI/CA split process.

6. A method according to Claim 1 or 2, comprising the additional step of:
writing said updated PB CI copy to a Shared Local Cache (SLC).

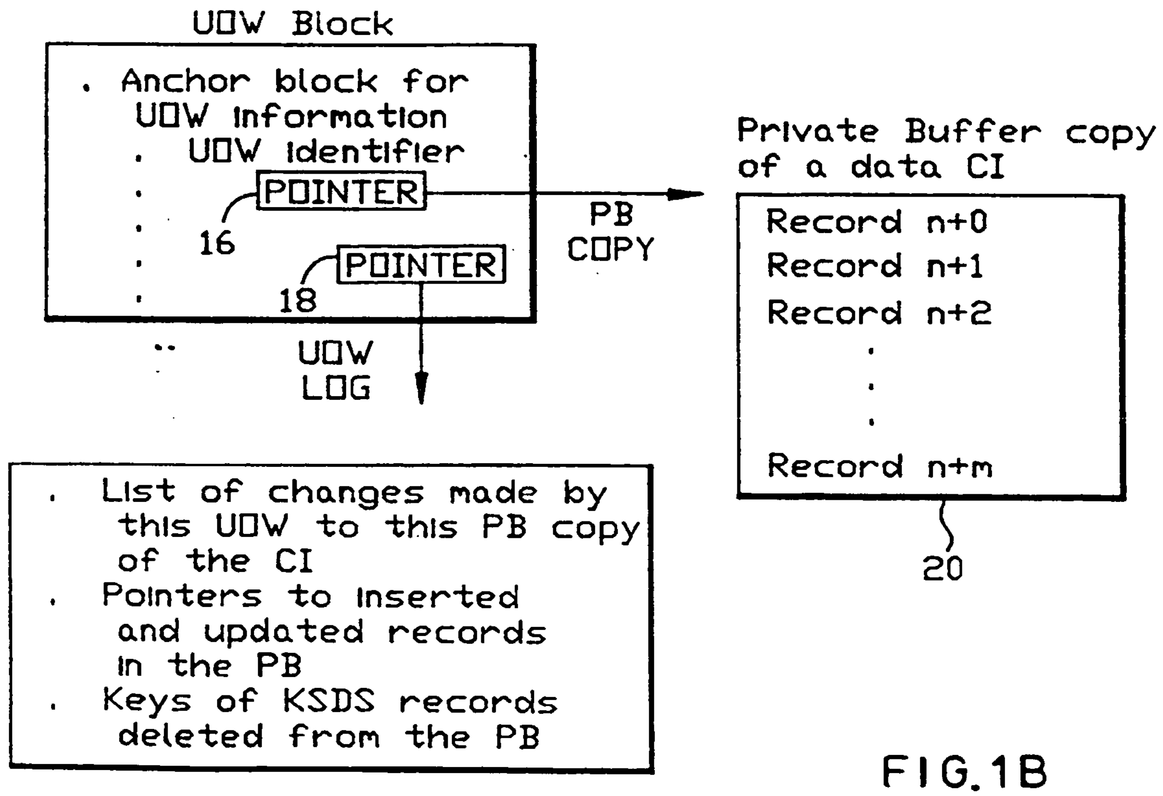
7. A method for record updating using solely record-level locking in a database system having a plurality of storage blocking intervals each including at least two records or the padded equivalent, the blocking intervals being accessed by a plurality of concurrent asynchronous processes through Shared External Storage (SES), the method comprising the steps of:
(a) ascertaining whether a first record in a predetermined blocking interval requested by a first process resides in the SES;
(b) fetching the most recent copy of the storage blocking interval containing the first record from the SES for exclusive access by the first process;
(c) changing the exclusive storage blocking interval copy by said first process and logging the changes;
(d) writing the changed exclusive storage blocking interval copy conditionally to SES, the private copy being accepted by SES if the storage blocking interval was not otherwise changed in SES by another process during the time since it was fetched by the first process; and
(e) if the changed exclusive storage blocking interval copy is not accepted by SES, fetching from SES the most recent storage blocking interval copy containing the first record and repeating steps (c) and (d).

8. A method according to Claim 7, comprising the additional step of:
granting an exclusive record-level lock to the first process on the first record.



SLC
SLC
SLC
SLC
SLC
12
CI
CI
CI
14
10
SES

FIG. 1A



UOW Block
. Anchor block for UOW information
. UOW identifier
POINTER
16
POINTER
18
PB COPY
UOW LOG
Private Buffer copy of a data CI
Record n+0
Record n+1
Record n+2
Record n+m
20
. List of changes made by this UOW to this PB copy of the CI
. Pointers to inserted and updated records in the PB
. Keys of KSDS records deleted from the PB

FIG.1B

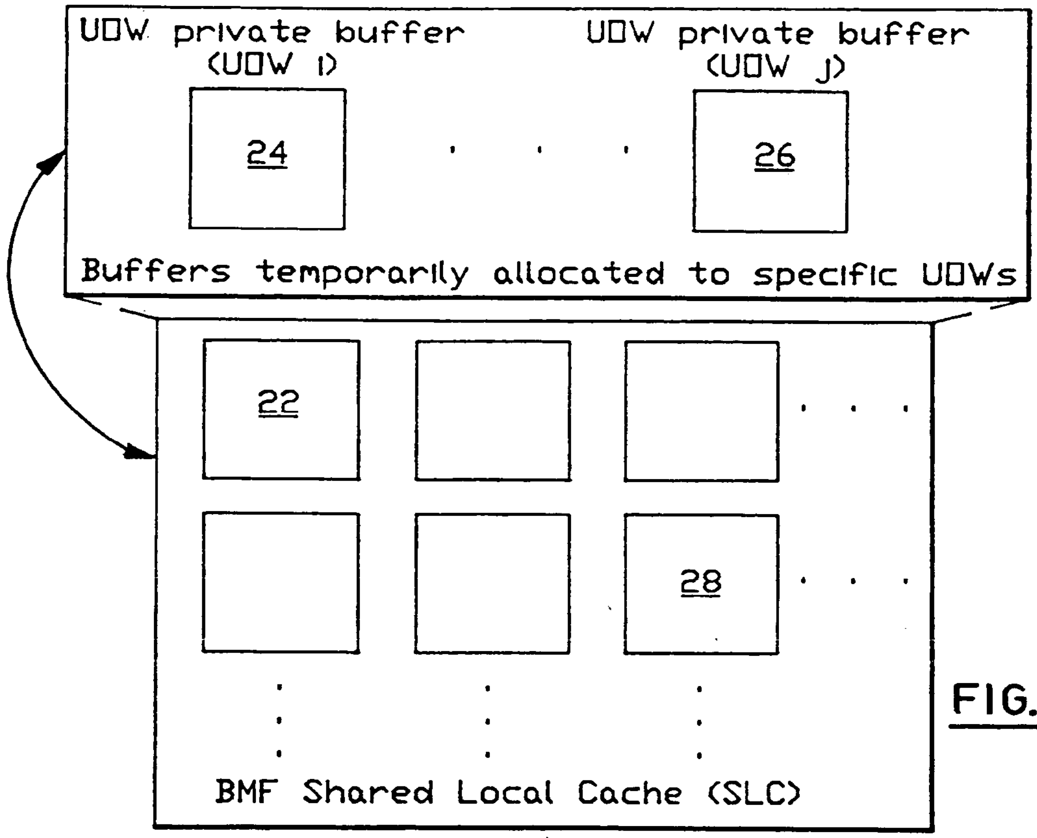

UOW private buffer (UOW I)
24
UOW private buffer (UOW J)
26
Buffers temporarily allocated to specific UOWs
22
28
BMF Shared Local Cache (SLC)

FIG. 2

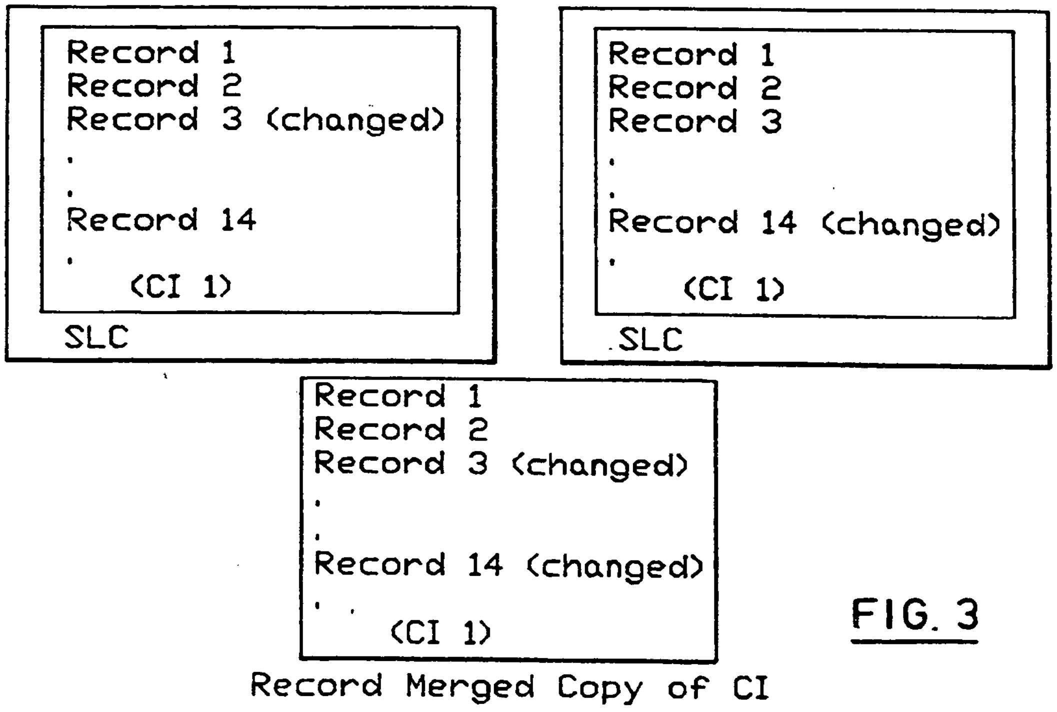

Record 1
Record 2
Record 3 (changed)
Record 14
(CI 1)
SLC
Record 1
Record 2
Record 3
Record 14 (changed)
(CI 1)
SLC
Record 1
Record 2
Record 3 (changed)
Record 14 (changed)
(CI 1)
Record Merged Copy of CI

FIG. 3

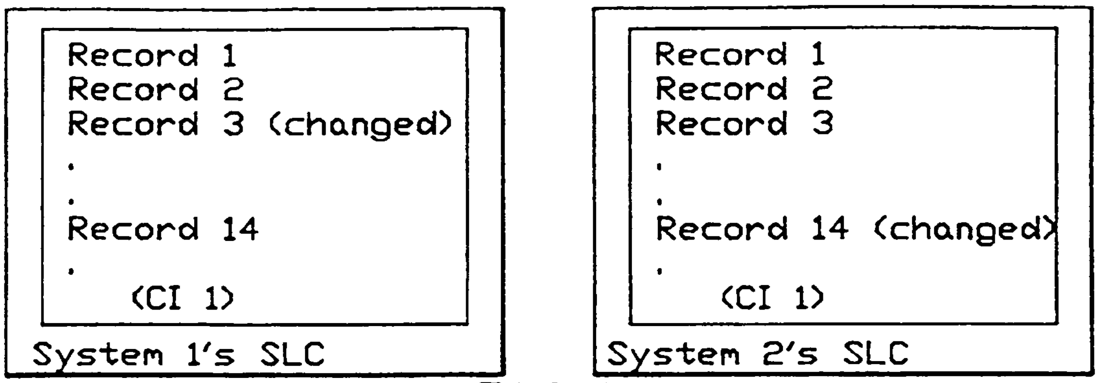


FIG. 4A

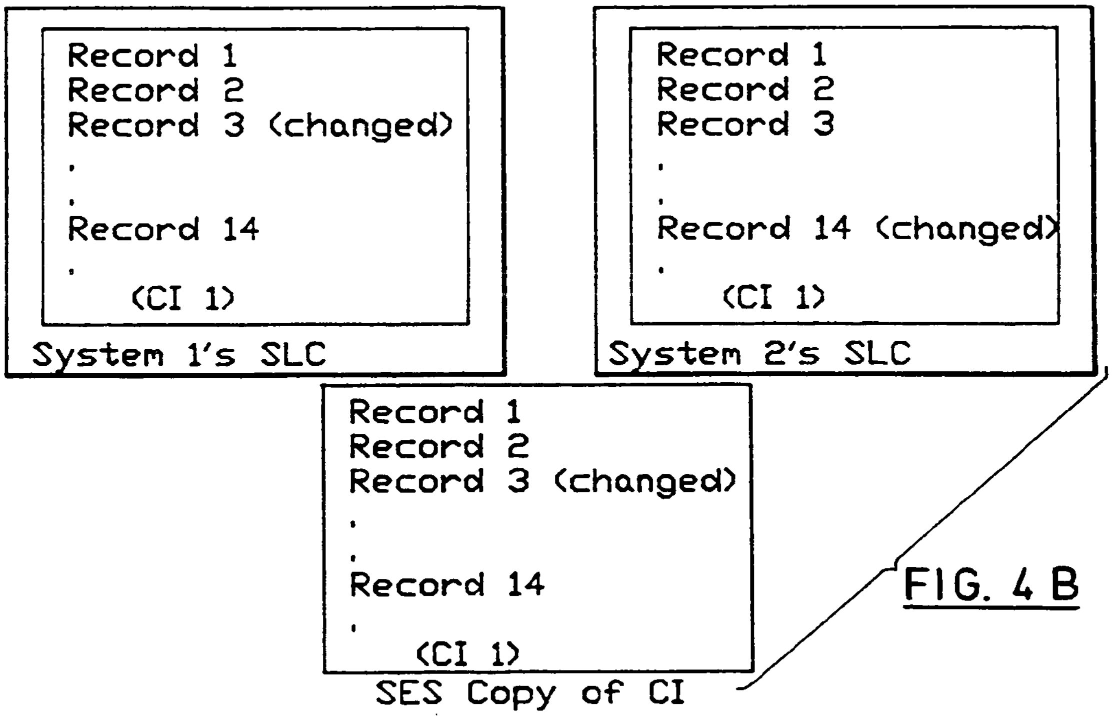


FIG. 4B

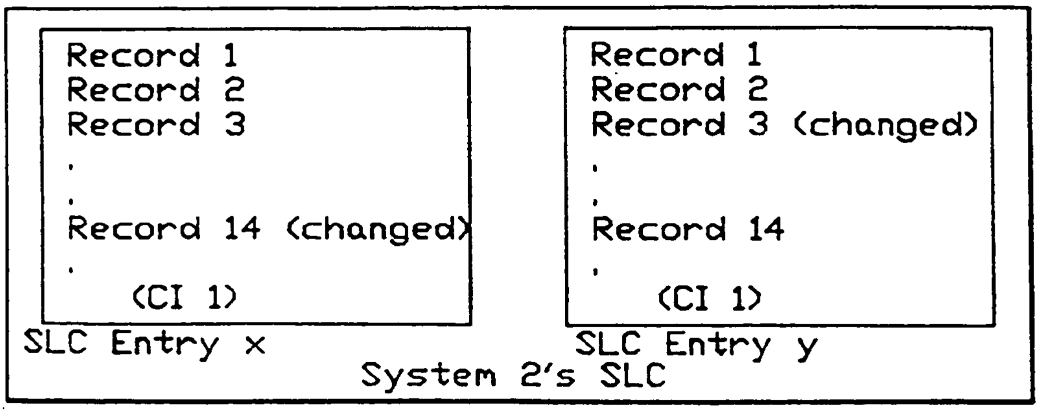


FIG. 4C

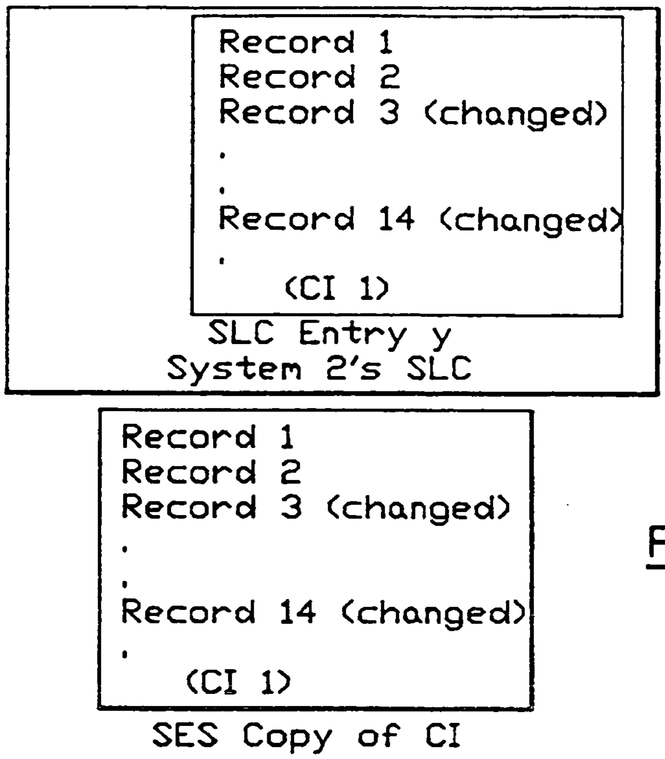

Record 1
Record 2
Record 3 (changed)
Record 14 (changed)
(CI 1)
SLC Entry y
System 2's SLC
Record 1
Record 2
Record 3 (changed)
Record 14 (changed)
(CI 1)
SES Copy of CI

FIG. 4D

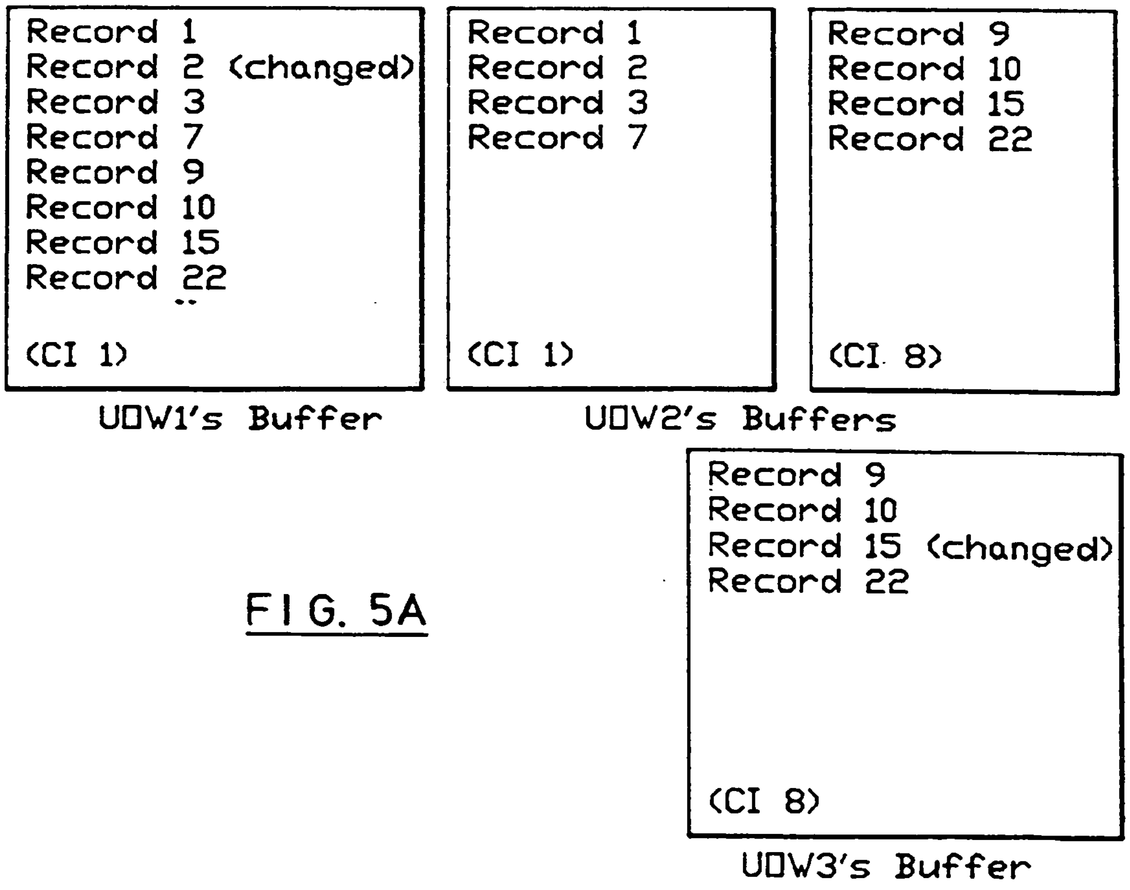

Record 1
Record 2 (changed)
Record 3
Record 7
Record 9
Record 10
Record 15
Record 22
(CI 1)
UOW1's Buffer
Record 1
Record 2
Record 3
Record 7
(CI 1)
Record 9
Record 10
Record 15
Record 22
(CI 8)
UOW2's Buffers
Record 9
Record 10
Record 15 (changed)
Record 22
(CI 8)
UOW3's Buffer

FIG. 5A

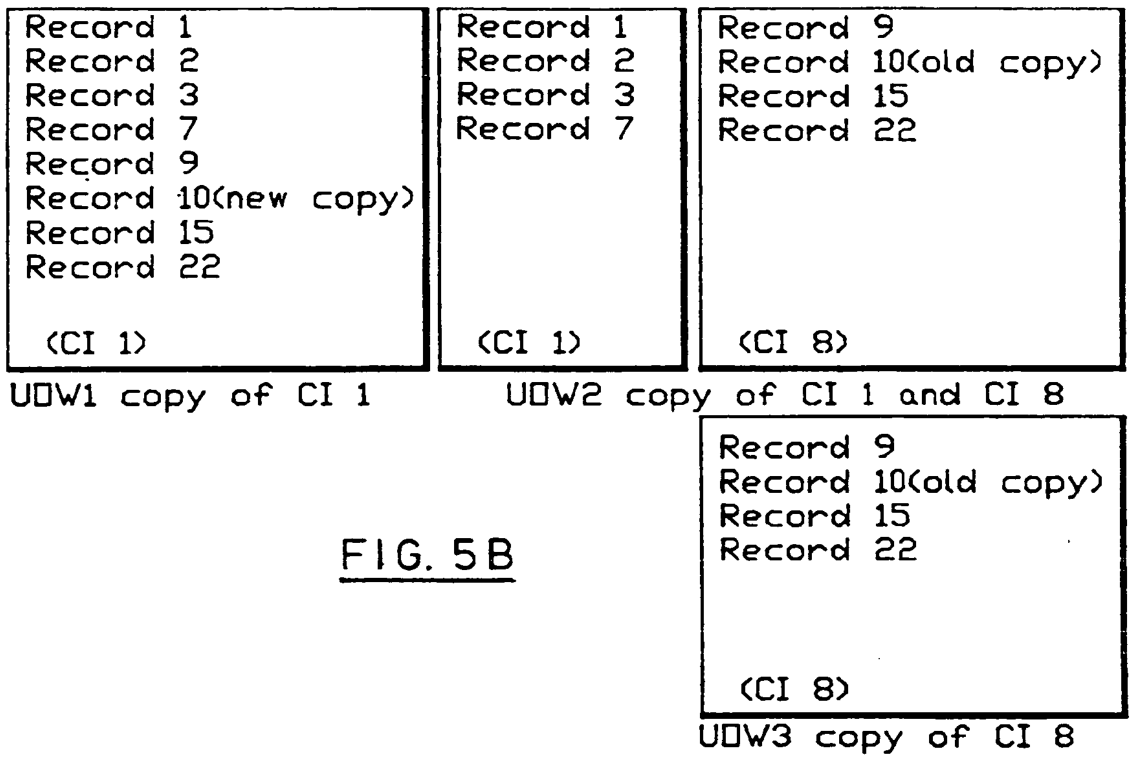

Record 1
Record 2
Record 3
Record 7
Record 9
Record 10(new copy)
Record 15
Record 22
(CI 1)
UOW1 copy of CI 1
Record 1
Record 2
Record 3
Record 7
(CI 1)
Record 9
Record 10(old copy)
Record 15
Record 22
(CI 8)
UOW2 copy of CI 1 and CI 8
Record 9
Record 10(old copy)
Record 15
Record 22
(CI 8)
UOW3 copy of CI 8

FIG. 5B

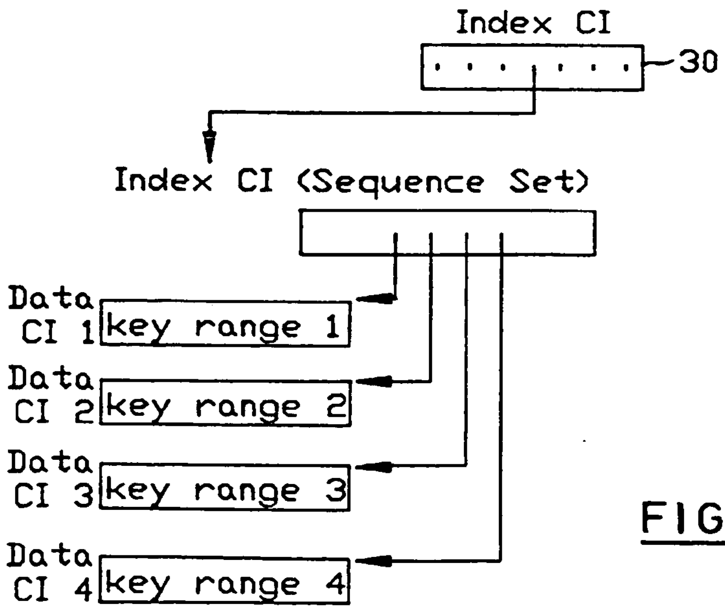

Index CI
30
Index CI (Sequence Set)
Data CI 1
key range 1
Data CI 2
key range 2
Data CI 3
key range 3
Data CI 4
key range 4

FIG. 6A

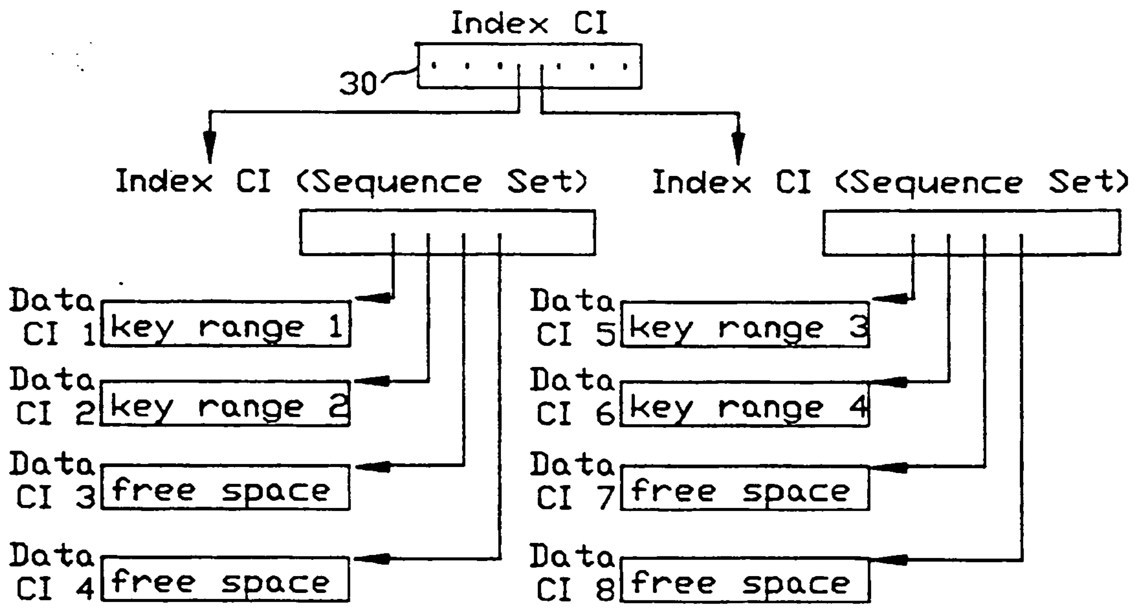


FIG. 6B

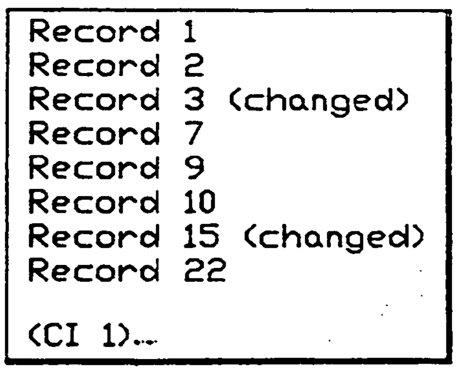


FIG. 7A

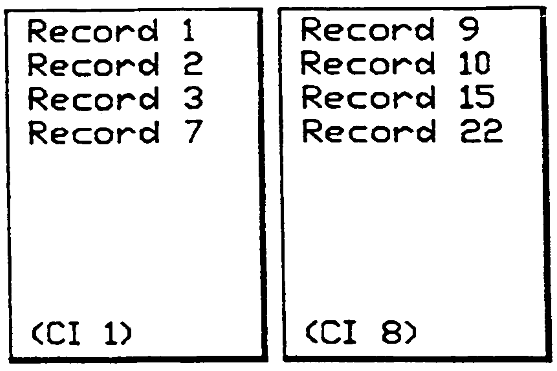


FIG. 7B

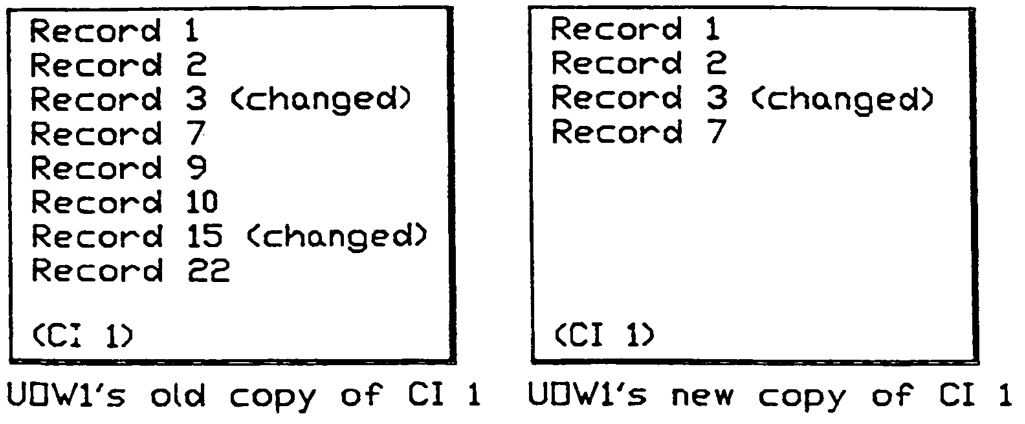


FIG. 7C

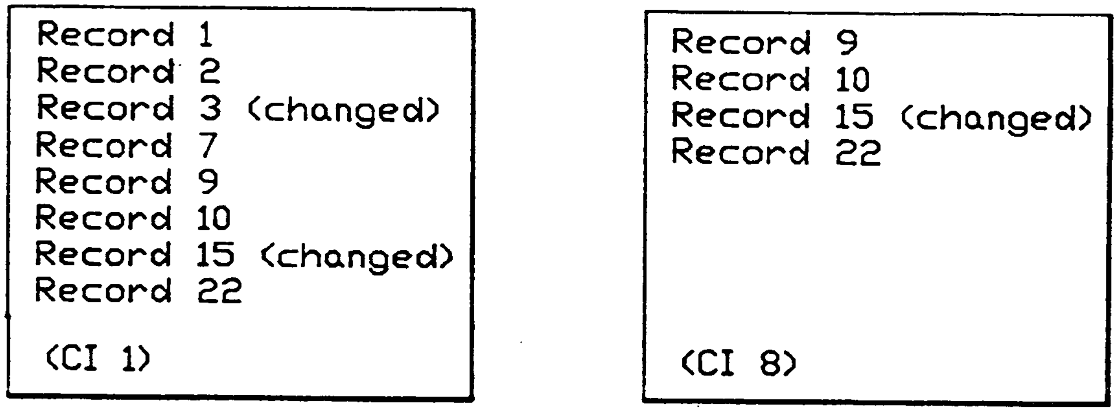

Record 1
Record 2
Record 3 (changed)
Record 7
Record 9
Record 10
Record 15 (changed)
Record 22
(CI 1)
UOW1's old copy of CI 1
Record 9
Record 10
Record 15 (changed)
Record 22
(CI 8)
UOW1's copy of CI 8

FIG. 7 D

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 549 140 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **92310801.3**

(22) Date of filing: **25.11.92**

(51) Int. Cl.^{5}: **G06F 15/403**

(30) Priority: **23.12.91 US 812677**

(43) Date of publication of application:
30.06.93 Bulletin 93/26

(84) Designated Contracting States:
DE FR GB

(88) Date of deferred publication of the search report:
03.11.93 Bulletin 93/44

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Strickland, Jimmy Paul**
18929 Alcott Way
Saratoga, California 95070(US)
Inventor: **Kapulka, Kenneth Michael**
6378 Felder Drive
San Jose, California 95123(US)

(74) Representative: **Burt, Roger James, Dr.**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) Record updating method.

(57) A Virtual Storage Access Management (VSAM) technique for maintaining data integrity of the record-level shared data in a multiprocessor system environment without the concurrency loss associated with locking at the data Control Interval (CI) or block level. Two or more concurrent processes can update different records within the same VSAM data CI using only a record-level lock. This feature relies upon multiple private data CI copies, Shared External Storage (SES), cache cross-invalidation, and a conditional writing operation. A first process updates a private copy of a data CI containing the target record from SES, logs the changes, and writes the changed data CI back to the SES conditioned upon no update having been made to the same block data CI by another process in the interim. If a second process has indeed updated the same data CI in the interim, the first process recycles by obtaining the most recent data CI copy from SES, combining it with the first process update log entries and attempting to again conditionally write the modified data CI to SES. This process of logging updates and spinning on an intervening update by another process preserves serialization by record-level locking alone.

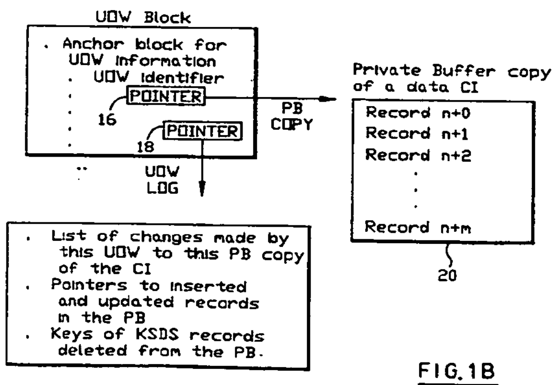

UOW Block
Anchor block for UOW information
UOW identifier
POINTER
16
POINTER
18
PB COPY
UOW LOG
Private Buffer copy of a data CI
Record n+0
Record n+1
Record n+2
Record n+m
20
List of changes made by this UOW to this PB copy of the CI
Pointers to inserted and updated records in the PB
Keys of KSDS records deleted from the PB.

FIG.1B

EP 0 549 140 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 92 31 0801

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X	EP-A-0 454 610 (IBM) * the whole document * ---	1-8	G06F15/403
X	EP-A-0 450 917 (IBM) * the whole document * ---	1-8	
A	IEEE PROCEEDINGS, SECOND INTERNATIONAL CONFERENCE ON COMPUTERS AND APPLICATIONS, CAT. NR. 87CH2433-1, 27 June 1987, BEIJING, CHINA pages 410 - 417 RAHM E. 'INTEGRATED SOLUTIONS TO CONCURRENCY CONTROL AND BUFFER INVALIDATION IN DATABASE SHARING SYSTEMS' -----	1,7	

TECHNICAL FIELDS SEARCHED (Int. Cl.5)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	08 SEPTEMBER 1993	SUENDERMANN R.O.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document